
b3-propagation

B3 Propagation is a specification for the header “b3” and those that start with “x-b3-”. These headers are used for trace context propagation across service boundaries.

Overview

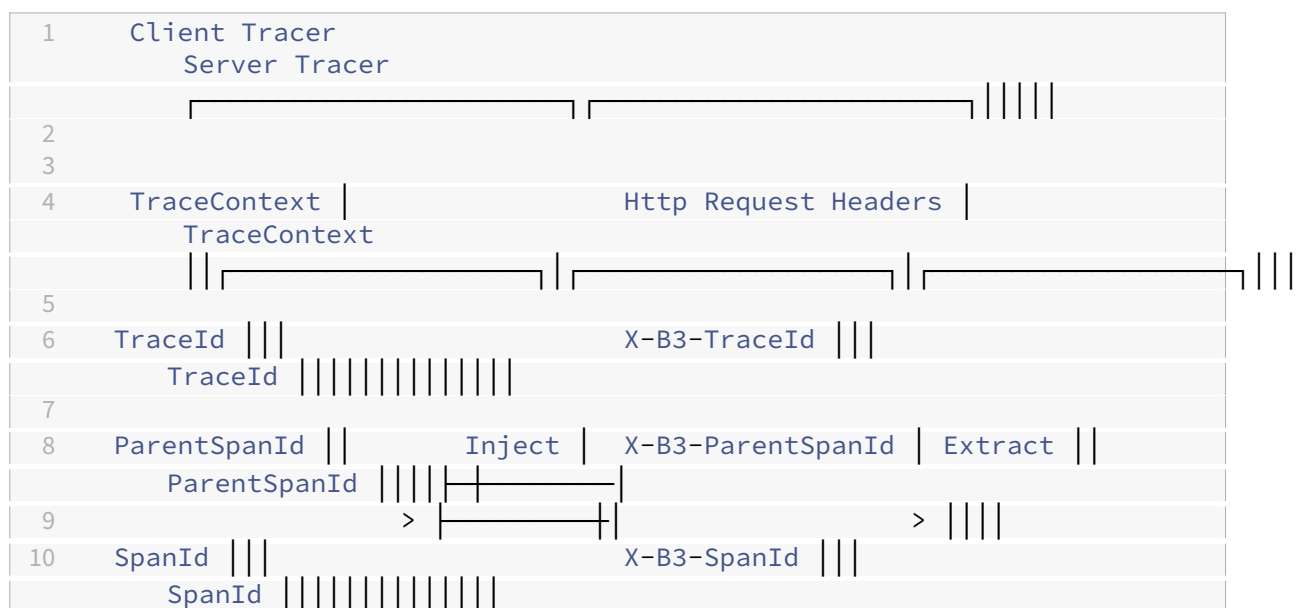
This specification elaborates identifiers used to place an operation in a trace tree. These attributes are propagated in-process, and eventually downstream (often via http headers), to ensure all activity originating from the same root are collected together. A sampling decision is made at the root of the trace, and this indicates if trace details should be collected and reported to the tracing system (usually Zipkin) or not.

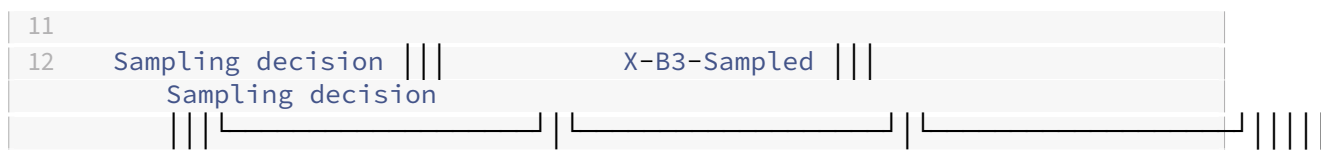
Overall Process

The most common propagation use case is to copy a trace context from a client sending an RPC request to a server receiving it.

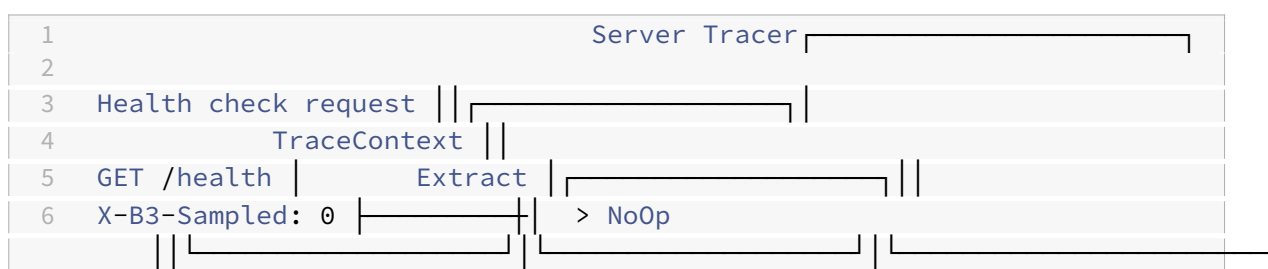
In this case, the same span ID is used, which means that both the client and server side of an operation end up in the same node in the trace tree.

Here’s an example flow using multiple header encoding, assuming an HTTP request carries the propagated trace:





Trace identifiers are often sent with a sampling decision. However, it is valid and common practice to send a sampling decision independently. Here's an example of a proxy forbidding tracing of a `/health` endpoint. The diagram shows the receiver making a NoOp trace context to ensure minimal overhead.



Identifiers

Trace identifiers are 64 or 128-bit, but all span identifiers within a trace are 64-bit. All identifiers are opaque.

Identifiers are almost always sent with Sampling state, but they can be sent alone to implement a “Defer” decision described below.

TraceId

The TraceId is 64 or 128-bit in length and indicates the overall ID of the trace. Every span in a trace shares this ID.

SpanId

The SpanId is 64-bit in length and indicates the position of the current operation in the trace tree. The value should not be interpreted: it may or may not be derived from the value of the TraceId.

ParentSpanId

The ParentSpanId is 64-bit in length and indicates the position of the parent operation in the trace tree. When the span is the root of the trace tree, there is no ParentSpanId.

Sampling State

Sampling is a mechanism to reduce the volume of data that ends up in the tracing system. In B3, sampling applies consistently per-trace: once the sampling decision is made, the same value should be consistently sent downstream. This means you will see all spans sharing a trace ID or none.

Here are the valid sampling states. Note they all applied to the trace ID, not the span ID: * **Defer**: aka I don't know yet! * **Defer** is used when trace identifiers are set by a proxy, but that proxy doesn't send data to Zipkin. The most common use case for deferring a decision is pre-provisioning trace identifiers. * **In all known encodings, defer is the absence of sampling state.** * **Deny**: aka don't sample or don't record * **Deny** is used to achieve a probabilistic rate or to prevent certain paths (such as health checks) from generating traces. Where possible, instrumentation should optimize deny such that less overhead occurs. * **Accept**: aka sample or record * **Accept** is used to achieve a probabilistic rate or to ensure certain paths (such as low-traffic endpoints) are always traced. When a trace is accepted, spans should be reported to zipkin except in overload scenarios. * **Debug**: aka force trace * **Debug** is a production troubleshooting aid used in tools like curl or chrome debug. **Debug** is an emphasized accept decision that implies accept, additionally reporting `Span.debug = true` for each span in the trace.

The most common use of sampling is probabilistic: eg, accept 0.01% of traces and deny the rest. **Debug** is the least common use case.

Sampling state is almost always sent with Identifiers, but it can be sent alone (a predefined decision). This is used for a few use cases: * **Deny**: Using a constant entry for a deny decision is more efficient than generating identifiers. Libraries that do this cannot do external ID correlation, though, for example, log correlation. * **Accept**: Some proxies send a constant entry for an accept decision as they want a specific path to be traced, but don't want to interfere with ID generation. * **Debug**: Some support guides send debug without IDs as it is easier to do in curl.

Encodings

Below are a number of predefined encodings. The oldest encoding is multiple HTTP headers (2012). Some application notes have been made since, as well a compact, single-header encoding. A binary

encoding may be defined as well.

Most users do not need to know the encodings listed here as they are built-in to libraries, frameworks and proxies. While this specification aims to be complete, it will not elaborate all use cases and practice. Please contact the Zipkin gitter if you are confused about something.

Custom Encodings

We recommend re-using existing encodings over developing new ones as there are usually subtleties that even experienced folks overlook. For example, one proxy accidentally chose to use a hyphen to substitute for an absent parent span ID. This caused broken traces, and in some extreme cases, crashed requests. Some types of software have multiple month or longer release cycles, which adds tension to add “spaghetti” to clean it up. Identifying and answering questions about such problems add support load unfairly to others as they weren’t consulted. Even ruling out format mismatches, misunderstanding of the feature set can cause expensive maintenance later.

The best way to avoid costly mistakes is to re-use an encoding. The second-best way is reading this spec completely when designing a bespoke encoding, and getting feedback from the authors on your work. If you are designing a new encoding, please contact the Zipkin gitter so that you have the best chance to avoid mistakes others made in the past.

Http Encodings

There are two encodings of B3: Single Header and Multiple Header. Multiple header encoding uses an `X-B3-` prefixed header per item in the trace context. Single header delimits the context into a single entry named `b3`. The single-header variant takes precedence over the multiple header one when extracting fields.

Multiple Headers

B3 attributes are most commonly propagated as multiple http headers. All B3 headers follows the convention of `X-B3-{name}` with special-casing for flags. When extracting state headers, the first value wins.

Note: Http headers are case-insensitive, but sometimes this encoding is used for other transports. When encoding in case-sensitive transports, prefer lowercase keys or the single header encoding which is explicitly lowercase.

As mentioned earlier, identifiers can be sent with or without sampling state and vice versa. It is important to understand the relationships between these headers. For example, `X-B3-TraceId` and `X-B3-SpanId` can be sent alone, or with a sampling header `X-B3-Sampled`. It is also valid to send sampling state independently, such as a deny decision `X-B3-Sampled: 0`.

Note: Two headers, `X-B3-Sampled` and `X-B3-ParentSpanId` can be absent. Absent means the header doesn't exist. An empty header (ex. `X-B3-Sampled:`) or an arbitrary nonsense value (ex `X-B3-ParentSpanId: -`) are examples of malformed data.

TraceId

The `X-B3-TraceId` header is encoded as 32 or 16 lower-hex characters. For example, a 128-bit TraceId header might look like: `X-B3-TraceId: 463ac35c9f6413ad48485a3953bb6124`. Unless propagating only the Sampling State, the `X-B3-TraceId` header is required.

SpanId

The `X-B3-SpanId` header is encoded as 16 lower-hex characters. For example, a SpanId header might look like: `X-B3-SpanId: a2fb4a1d1a96d312`. Unless propagating only the Sampling State, the `X-B3-SpanId` header is required.

ParentSpanId

The `X-B3-ParentSpanId` header may be present on a child span and must be absent on the root span. It is encoded as 16 lower-hex characters. For example, a ParentSpanId header might look like: `X-B3-ParentSpanId: 0020000000000001`

Sampling State

An accept sampling decision is encoded as `X-B3-Sampled: 1` and a deny as `X-B3-Sampled: 0`. Absent means defer the decision to the receiver of this header. For example, a Sampled header might look like: `X-B3-Sampled: 1`.

Note: Before this specification was written, some tracers propagated `X-B3-Sampled` as **true** or **false** as opposed to 1 or 0. While you shouldn't encode `X-B3-Sampled` as **true** or **false**, a lenient implementation may accept them.

Debug Flag Debug is encoded as `X-B3-Flags: 1`. Absent or any other value can be ignored. Debug implies an accept decision, so don't also send the `X-B3-Sampled` header.

Single Header

A single header named `b3` standardized in late 2018 for use in JMS and w3c `tracestate`. Design and rationale are captured here. Check or update our status page for adoption status.

In simplest terms `b3` maps propagation fields into a hyphen delimited string.

`b3={TraceId}-{SpanId}-{SamplingState}-{ParentSpanId}`, where the last two fields are optional.

For example, the following state encoded in multiple headers:

```
1 X-B3-TraceId: 80f198ee56343ba864fe8b2a57d3eff7
2 X-B3-ParentSpanId: 05e3ac9a4f6e3b90
3 X-B3-SpanId: e457b5a2e4d86bd1
4 X-B3-Sampled: 1
```

Becomes one `b3` header, for example:

```
1 b3: 80f198ee56343ba864fe8b2a57d3eff7-e457b5a2e4d86bd1-1-05
   e3ac9a4f6e3b90
```

Note: When only propagating a sampling decision, the header is still named `b3`, but only contains the sampling state.

A deny decision encodes as:

```
1 b3: 0
```

TraceId

The first position of the `b3` value is the 32 or 16 lower-hex character `TraceId`. Unless propagating only the Sampling State, the `TraceId` is required.

For example, a `TraceId` of `80f198ee56343ba864fe8b2a57d3eff7` encodes in the left-most position:

```
1 b3: 80f198ee56343ba864fe8b2a57d3eff7-e457b5a2e4d86bd1-1
```

SpanId

The second position of the **b3** value is the 16 lower-hex character SpanId. Unless propagating only the Sampling State, the SpanId is required.

For example, a SpanId of **e457b5a2e4d86bd1** encodes after the left-most hyphen:

```
1 b3: 80f198ee56343ba864fe8b2a57d3eff7-e457b5a2e4d86bd1-1
```

Sampling State

When the sampling state isn't Defer, it is encoded as a single hex character in the third position of the **b3** value. Sampling state can also be sent on its own (with no identifiers).

Sampling State is encoded as a single hex character for all states except Defer. Defer is absence of the sampling field. * Accept: 1 * Deny: 0 * Debug: d

For example, a debug trace could look like this

```
1 b3: 80f198ee56343ba864fe8b2a57d3eff7-e457b5a2e4d86bd1-d
```

A deny decision may omit identifiers and look like this:

```
1 b3: 0
```

ParentSpanId

When present, ParentSpanId is the 16 lower-hex characters in the final position of **b3**.

For example, a ParentSpanId of **05e3ac9a4f6e3b90** encodes in the right-most position:

```
1 b3: 80f198ee56343ba864fe8b2a57d3eff7-e457b5a2e4d86bd1-1-05
    e3ac9a4f6e3b90
```

gRPC Encoding

B3 attributes can also be propagated as ASCII headers in the Custom Metadata of a request. The encoding is exactly the same as Http headers, except the names are explicitly or implicitly down-cased.

For example, when using multiple headers, **X-B3-ParentSpanId: 0020000000000001** encodes as an ASCII header **x-b3-parentspanid** with the same value.

JMS Encoding

JMS (Java Message Service) has constraints that disallow `X-B3-` prefixed headers. As such, only the single header format `b3` should be used with JMS. As messaging spans never share a `SpanId`, it is encouraged to omit the `ParentSpanId` field when propagating.

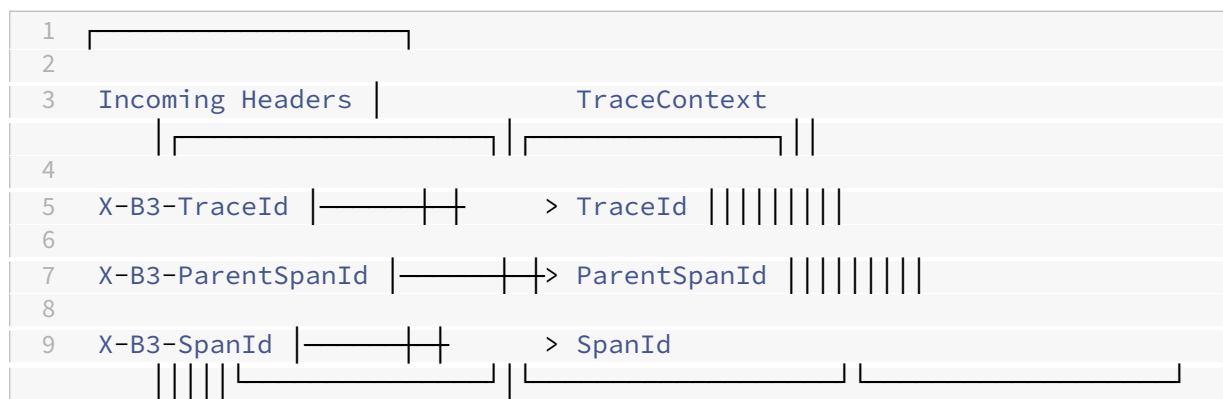
Refer to our design documentation for details.

Frequently Asked Questions

Why is `ParentSpanId` propagated?

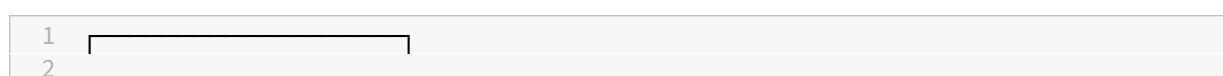
In B3, the trace context is extracted from incoming headers. Timing and metadata for the client and server side of an operation are recorded with the same context. The `ParentSpanId` is the ID of the operation that caused the current RPC. For example, it could be the ID of another server request or a scheduled job. `ParentSpanId` is propagated so that when data is reported to Zipkin, it can be placed in the correct spot in the trace tree.

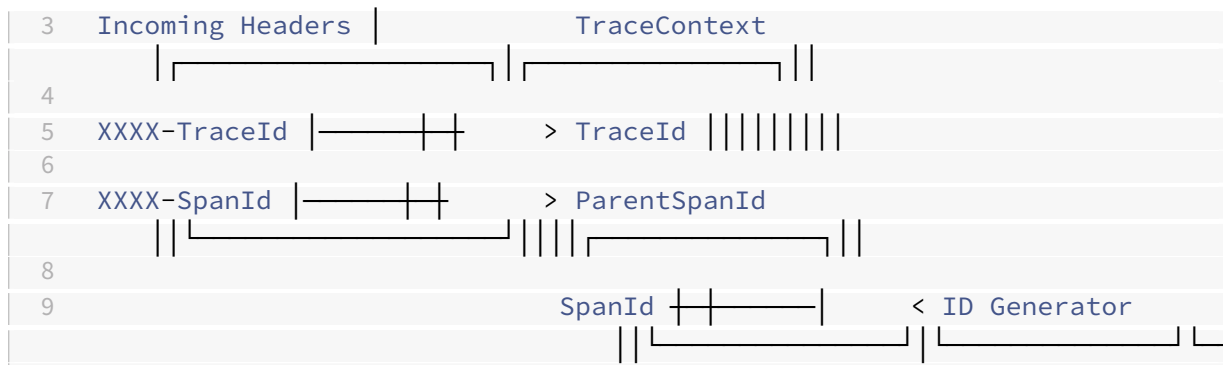
Here's an example of a B3 library extracting a trace context from incoming http request headers:



Some propagation formats look similar to B3, but don't propagate a field named `parent`. Instead, they propagate a span ID field which serves the same purpose as `ParentSpanId`. Unlike B3, these systems use a different span ID for the client and server side of an RPC. When a server reads headers like this, it is expected to provision a new span ID for itself, and use the one it extracted as its parent.

Here's an example of an alternate library composing a trace context with incoming http request headers and an ID generator:





In both B3 and the above example, incoming headers contain the parent's span ID, and three IDs (trace, parent, span) end up in the trace context. The difference is that B3 uses the same span ID for the client and server side of an RPC, where the latter does not.

Why propagate a reject sampling decision?

It may not be obvious why you'd send a reject sampling decision to the next hop. Imagine a service decides not to trace an operation and makes 2 out-going calls, and these branched out further. If reject ("don't trace") isn't propagated, the system might receive only parts of the operation, confusing users.

Another reason to reject sampling is to prevent overwhelming the collector with high throughput. If the reject-sampling decision is not propagated then there's no way to tell recipients of outbound calls that they should not send span info to the collector.

Why send trace IDs with a reject sampling decision?

While valid to propagate only a reject sampling decision (`X-B3-Sampled: 0`), if trace identifiers are established, they should be propagated, too. The tracing system is often not the only consumer of trace identifiers. Services may still want to do things with the tracing info even if they aren't sending span data to the collector, e.g. tag logs with trace IDs.

Why defer a sampling decision?

Deferring the sampling decision is special-case. The only known use-cases are the following:

- Debug trace: A debug trace is implicitly an accepted trace
- Externally provisioned IDs: When you want to control IDs, but not sampling policy

Unless it is a debug trace, leaving out a sampling decision is typically ID correlation. For example, someone re-uses a global identifier from another system, or correlating in logs. In these cases, the caller knows the ID they want, but allows the next hop to decide if it will be traced or not. The caller should not report a span to the tracing system using this ID unless they propagate an accept sampling decision.

Why is Debug encoded as X-B3-Flags: 1?

The first tracer to use B3 was Finagle. In Finagle, thrift-rpc was a dominant protocol, and a fixed-width binary encoding held B3 attributes. This binary encoding used a bit field to encode the sampling decision and the debug flag.

Http encoding chose to use a separate header for the sampling decision, but kept the flags field for debug. In hind sight, it would have made more sense to use a separate header such as `X-B3-Debug: 1` as no other flags were added and no implementation treats `X-B3-Flags` as a bit field. That said, renaming the field would cause more damage than good, so it was left alone.

What is the relationship between B3 context and trace data formats?

When you see B3 headers of any kind, you cannot assume the out-of-band data is in Zipkin format, or even that data is sent to a tracing system. For example, some embed trace data into log lines correlated with the same IDs in the headers. Others use B3 headers for dramatically different trace formats.

That said, it is typically the case that data is sent out-of-band, and outgoing fields match some fields in that data. For example, in Zipkin format, all fields except Accept and Deny flags are reported. That's because Accept is implied and Deny would not result in reported data.

Handling B3 when reporting data to a tracing system

It is possible that a proxy, such as Envoy, accepts trace data correlated with B3 headers sent by the application. Depending on configuration, that proxy could start a trace for the act of sending trace data, amplifying traffic to your tracing system with low or negative value traces. While most sites will not trace data sent to a tracing system, there are generally 2 ways to address this proxy setup:

1. Configure the proxy to not trace your trace reporting endpoint (Ex in Zipkin, `POST /api/v2/spans`)
2. Send `b3: 0` header (an explicit Deny), to prevent tracing.

We recommend having your trace reporting library send `b3: 0`. This cements a cost to everyone, only to help those proxying their trace data. However, it is becoming common in mesh environments to proxy everything, and in reality there is limited ability for end users to change the sampling policy of proxies.

Note: we don't suggest sending also `x-b3-sampled: 0` as the proxy problem is recent and largely contained to Envoy, which already supports B3 single format. Sending a larger second header would increase the pain to others who don't have this problem anyway. Put another way, the few proxies missing B3 single header support should update instead of burdening more people from lack thereof.